Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science and Engineering

Bachelor's Project

Convergence of UCT in games with imperfect information

Marián Briedoň

Supervisor: Mgr. Viliam Lisý, PhD.

Study Programme: Open Informatics, Bachelor

Field of Study: Informatics and Computer Science

May 21, 2015

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Marián B r i e d o ň

Studijní program: Otevřená informatika (bakalářský)

Obor: Informatika a počítačové vědy

Název tématu: Konvergence UCT ve hrách s nedokonalou informací

Pokyny pro vypracování:

UCT je Monte Carlo algoritmus prohledávání herního stromu (MCTS), který byl v poslední době úspěšně aplikován na řešení her s dokonalou informací [1]. Posléze byly algoritmy na bázi UCT použity i pro hry s nedokonalou informací (např. [2,3]). Přímočaré použití těchto algoritmů ale může konvergovat k nesprávným řešením i pro velmi jednoduché hry [4]. Na druhé straně, tyto algoritmy často v praxi hrají velmi dobře [5]. V této práci student:

1. vytvoří přehled existující literatury ohledně použití MCTS a hlavně UCT ve hrách s nedokonalou informací,
2. upraví nebo naimplementuje vybrané algoritmy v existujícím softwarovém balíku algoritmů pro teorii her,
3. experimentálně nebo formálně prozkoumá konvergenci těchto algoritmů v hrách různé složitosti (maticové hry, hry se současnými tahy, hry v extenzivní formě) aby popsal, k jakým řešením mohou tyto algoritmy konvergovat,
4. identifikuje třídy her nebo jednoduché příklady her, v kterých tyto algoritmy zpravidla konvergují nebo nekonvergují k Nashově rovnováze,
5. případně navrhne vylepšení zkoumaných algoritmů a vyhodnotí jejich přínos.

Seznam odborné literatury:

[1] Browne, C. B., Powley, E., Whitehouse, et al. "A survey of monte carlo tree search methods." Computational Intelligence and AI in Games, IEEE Transactions on 4.1 (2012): 1-43
[2] Cowling, P. I., Powley, E. J., & Whitehouse, D. (2012). Information set Monte Carlo tree search. Computational Intelligence and AI in Games, IEEE Transactions on, 4(2), 120-143.
[3] Ponsen, M. J., De Jong, S., & Lanctot, M. (2011). Computing Approximate Nash Equilibria and Robust Best-Responses Using Sampling. J. Artif. Intell. Res.(JAIR), 42, 575-605.
[4] Shafiei, M., Sturtevant, N., & Schaeffer, J. (2009). Comparing UCT versus CFR in simultaneous games. Proceedings of the IJCAI Workshop on General Intelligence in Game-Playing Agents (GIGA) (pp. 75–82).
[5] Lisý, V. (2014). Monte Carlo Tree Search in Imperfect-Information Games. PhD Thesis. Czech Technical University in Prague.

Vedoucí bakalářské práce: Mgr. Viliam Lisý, MSc.

Platnost zadání: do konce letního semestru 2015/2016

L.S.

doc. Dr. Ing. Jan Kybic
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 14. 1. 2015

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

BACHELOR PROJECT ASSIGNMENT

Student: Marián B r i e d o ň

Study programme: Open Informatics

Specialisation: Computer and Information Science

Title of Bachelor Project: Convergence of UCT in Imperfect Information Games

Guidelines:

UCT is a Monte Carlo tree search (MCTS) algorithm that has recently been very successful in solving perfect information games [1]. Following this success, UCT-based algorithms have also been used for imperfect information games (e.g., [2,3]). The straightforward application of this algorithm seems to converge to a wrong solution even in very simple games [4]. On the other hand, it often produces empirically strong game play [5].
In this work, the student will:

1. review the existing work in applying MCTS and mainly UCT in imperfect information games;
2. adapt or implement selected algorithms in an existing game theory software package;
3. empirically or formally analyze convergence of these algorithms in imperfect information games of various complexity (i.e., matrix games, simultaneous move games, extensive form games) to identify the solutions to which the algorithms can converge;
4. identify classes of games or simple example games that allow or prevent convergence of these algorithms to Nash equilibrium;
5. optionally, propose improvements of the algorithms and evaluate their effectiveness.

Bibliography/Sources:

[1] Browne, C. B., Powley, E., Whitehouse, et al. "A survey of monte carlo tree search methods." Computational Intelligence and AI in Games, IEEE Transactions on 4.1 (2012): 1-43
[2] Cowling, P. I., Powley, E. J., & Whitehouse, D. (2012). Information set Monte Carlo tree search. Computational Intelligence and AI in Games, IEEE Transactions on, 4(2), 120-143.
[3] Ponsen, M. J., De Jong, S., & Lanctot, M. (2011). Computing Approximate Nash Equilibria and Robust Best-Responses Using Sampling. J. Artif. Intell. Res.(JAIR), 42, 575-605.
[4] Shafiei, M., Sturtevant, N., & Schaeffer, J. (2009). Comparing UCT versus CFR in simultaneous games. Proceedings of the IJCAI Workshop on General Intelligence in Game-Playing Agents (GIGA) (pp. 75–82).
[5] Lisý, V. (2014). Monte Carlo Tree Search in Imperfect-Information Games. PhD Thesis. Czech Technical University in Prague.

Bachelor Project Supervisor: Mgr. Viliam Lisý, MSc.

Valid until: the end of the summer semester of academic year 2015/2016

L.S.

doc. Dr. Ing. Jan Kybic
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, January 14, 2015

Aknowledgements

I would like to thank all the people who contributed in some way to the work described in this thesis. I would like to specially thank Viliam Lisý for support and for leading me through the work. I also thank my parents.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 21. 5. 2015 .....

Abstract

This thesis deals with the Upper confidence Tree algorithm, which belongs to the big family of the Monte Carlo Tree Search algorithms. We focused on two-player zero-sum games. We observed properties of UCT to find out, why it converges so fast and when UCT doesn't converge to the Nash equilibria. We empirically analyzed a convergence of UCT in simple matrix games and sequential simultaneous move games of various complexity. We used randomly generated games for finding problems of a covergence of UCT. We directly compared two modifications, namely the sliding window and the non-deterministic modification. Both of them have their pros and cons. We propose an improvement of the non-deterministic modification.

Abstrakt

Táto práca sa zaoberá algoritmom the Upper confidence Tree, v skratke UCT, ktorý patrí do veľkej rodiny the Monte Carlo Tree Search algoritmov. Zamerali sme sa na zero-sum hry pre dvoch hráčov. Pozorovali sme vlastnosti UCT, aby sme zistili, prečo konverguje tak rýchlo a kedy UCT nekonverguje k Nashovmu equilibriu. Empiricky sme analyzovali konvergenciu UCT v jednoduchých maticových hráča v sekvenčných hrách so simultánnymi ťahmi rôznej komplexnosti. Použili sme náhodne generované hry na hľadanie problémov konvergencie UCT. Priamo sme porovnali dve modifikácie, menovite Sliding Window a nedeterministickú modifikáciu. Obidve majú svoje plusy a mínusy. Navrhli sme zlepšenie nedeterministickej modifikácie.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Upper Confidence Tree search(UCT) is an algorithm used for finding strategies in games. It has strong experimental results, for an instance in the game GO. There are a lot of studies concerning UCT, but I didn't find a theoretical study about UCT in an imperfect-information game. Also I was unable to find studies concerning empirical convergence to the Nash equilibria. Studies usually compare UCT with the exponential exploration and exploitation(Exp-3)[13], but they don't compare a UCT modifications. We decided to compare a two modifications of the UCT. The modifications are the sliding window and the non-deterministic variant. The reason why we chose these two modifications is that they are most used and to know which modification is better. We tested both modifications for the purpose of comparing them. We found flows in both modifications.

UCT has a problem with a synchronization[12]. This problem is well known. Our idea was to find another problem of UCT. We found games in which we can identify the synchronization problem, which helps us to distinguish games with other problems. A problem we found was in an initial uniform distribution of actions.

We observe UCT's behaviour and explore its properties. First we observed its behaviour in dominated strategies. It allowed us to observe the rate of changing counted strategies in UCT. For the purpose of determining the dominance, we created a linear program for finding dominated and iteratively dominated strategies. We iteratively increased simulations in observation and stored strategies counted in each observation. We observe what effect have the dominance on UCT. Than we will test UCT on randomly generated games.

We focused on the convergence of UCT. We tried a more complex way of an approach. We create all variations of 3*3 two-player simultaneous zero-sum games with payoffs of -1,0,1. Together we tested 19680 games. After the test, we processed the obtained data into a graph of an exploitable distribution over these games and a graph of a percentage of changing strategies in each step. Which we then compare to a graph of a 2,000 randomly generated games.

The games were 4 actions per player zero-sum games with payoffs from 8 to -8. After thousands of games tested we found interesting games. We found a simple matrix game in which UCT didn't converge to the Nash equilibria. We continue our experimentation by creating randomly generated sequential simultaneous move games. On this variety of games we compared the nondeterministic modification of UCT and the Sliding Window. We have some ideas of the improvements of the modifications of the UCT.

Chapter 2

Game Theory

2.1 Introduction

The Game Theory is about the interaction of agents. If we make a reasonable assumption that agents will always have some preferences, we can then create utility functions for each agent. Let's assume that all agents want to maximize their utility. So the game describes interactions of agents, who choose an action, with a purpose of maximizing their a utility outcome. So these agents are rational and try to optimize their behaviour according to utility values. A game describes possible actions and the utility functions of players (agents). A game can also be viewed as an optimization problem. This allows us to see that the games can be easily written as a stochastic optimization problem and solved by stochastic programs.

In the Game Theory, there exists more than one way to represent a game. We choose to focus on two-player zero-sum games. These games are best described by these two representations.

- **Normal-form games**
  - an utilities are written as matrices
  - easy to find an utility outcome of players actions
  - hard to describe probabilities
  - best used for two player games
- **Extensive-form games**
  - an utilities are written as tree
  - harder to find an utility outcome of an players actions
  - easy to describe probabilities
  - used by the Monte Carlo Tree Search

2.2 Normal-form games

2.2.1 Normal-form games

This representation is fundamental because most of the other representations can be reduced to a normal-form game. This means we can compare any two games, in this representation. A game written in this form represents every state of the world. Which can be good for getting a complete image of a game, but it increases the size of a representation. This mean we directly see the result of every action of every player.

A (finite, n-person) normal-form game is a tuple (N,A, u), where:

- N is a finite set of n players, indexed by i
- A = A_1 x $\cdots$ x A_n, where A_i is a finite set of actions available to player i. Each vector $a = (a_1, ..., a_n) \in$ A is called an **action profile**
- $u = (u_1, ..., u_n)$ where $u_i : A \to R$ is a real-valued utility function for player i

  In the Figure 2.1 we can see a normal-form game. The rows represent actions of the first player and the columns actions of the second player. At the picture, we can see the Rock, Scissors, and Paper game. If the first player choose to play the paper and the second player chooses to play the rock action, the result is in the second row and the first column.

	Rock	Paper	Scissors
Rock	$0, 0$	$-1, 1$	$1, -1$
Paper	$1, -1$	$0, 0$	$-1, 1$
Scissors	$-1, 1$	$1, -1$	$0, 0$

Figure 2.1: Rock, Paper and Scissor as normal-form game

2.2.2 Strategies in Normal-form games

We defined a game, with a set of actions available to players and now we are going to define a set of strategies. A simple type of a strategy is to choose an action and play it every time. This is called the **pure strategy**. Every strategy has an **expected utility** it is an average utility for a strategy. In a Normal-form game, given a strategy $s = \{s_1, s_2, ..., s_n\}$:

$$u(s) = \sum_{i=1}^{n} u(i) * s(i) \tag{2.1}$$

More complicated type of a strategy is a **mixed strategy**. It randomizes actions of a player using a probability distribution. This allows to randomize player decisions and let him increase an expected utility from a strategy, which he is playing. The set of strategy profiles is a Cartesian product of the individual mixed strategy.[14]

A support of a mixed strategy is the set of players pure strategies, which have a non-zero probability being played. A mixed strategy in the Nash equilibria has equal an expected utility in a support. That means all actions(pure strategies) used in a mixed strategy have same expected utility.

A mixed strategy profile is the Cartesian product of every mixed strategy set $S_1 x ... x S_n$. That means that it includes precisely a one strategy for every player.

2.2.3 Nash equilibria

We can compare strategies using an expected utility. A strategy with a higher expected utility is considered better than the one with the lower expected utility. Now we have a method to compare strategies. **The best response** is a strategy with the biggest expected utility with a respect to an opponent strategy. The best response is not unique, only in a special cases, when it is the pure strategy.

If every player plays a best response to all other players strategies, then a strategy profile is called the **Nash equilibria**. **An exploitability** is a difference between the best response to a strategy and a game value. In a two-player zero-sum games, it can be counted as a half of the sum of the best response to a players strategies.[13]

2.2.4 Domination

In the Normal-form, it is easy to observe a **Domination** of a strategy [14]. A strategy is dominated if it has all actions with the same or a lower expected utility than any combinations of other strategies. This means in all actions of a dominant strategy has the same expected values as all other strategies. We can differ a two types of a dominance. A week domination allows to have the same value. A strong dominance requires all actions to have a higher expected utility. We can reduce a game by removing dominated strategies because it's not convenient to play them for agents.

Removing dominated strategies for all players in a game creates a smaller version of the same game. If there is a possibility to reduce a game again, then we call these dominated strategies the **Iteratively dominated strategies**. Using this method we can iteratively remove dominated strategies and gain a much smaller game, but with the same Nash equilibrium as the original one.

2.2.5 Zero-sum games

It is the special type of games, mostly used for two player games. In these games, the total utility distributed among the players doesn't change during the game. In every state of a game, one player gains the same amount of a utility as all combined opponents loose. This

allows that all Nash equilibria strategy profiles are interchangeable. That means it doesn't matter which the Nash equilibria is the player playing.

We can define the **Game Value** which is the expected value of a player played strategies in the Nash equilibrium. If one player is playing the Nash equilibrium and an other players play any combination of actions in a support of the Nash equilibria strategy, he get the Game value[14].

2.3 Extensive-form games

The normal-form game doesn't have incorporated any sequence of the player actions[14, p. 117]. This flow is solved in an alternative form. The **Extensive-form game** is a representation of games using a tree. Using a tree as representation allows to see the sequence of actions in a game. Adding an action in the extensive-form game, will add one new edge and one new node. Adding the same action to the normal-form game will increase the matrix size, which result in a higher memory requirements than the same game represented as an extensive-form game. A tree also allows us to see an impact of each action on the game, which will be very difficult in a normal-form representation. The one downfall of this representation is that is hard to see a domination of strategies and is harder to calculate the Nash equilibrium than in the normal-form game.[9]

We can divide this form into a two cases:

- perfect information games
  - all players know in which node they are
  - player know all actions of the other players
  - no sets
- imperfect information games
  - player don't necessary know opponent actions
  - states are grouped into sets
  - player can't distinguish states in sets

2.3.1 The Games with the perfect-information

The games with the perfect information are represented as a tree. Each node represents a choice of each player, each edge represents a possible action and leaves represents a final outcome of a game. A root is the start of a game and leaves are the end of a game.

We can define a **History**, which is a sequence of actions from a root leading to the node. Using a graph theory, each node create a sub-tree, which is called the **Descendants of node**. From a history and descendants, we observe an actions sequence which led to this node and all possible outcomes of this particular node.

A pure strategy in an extensive-form with a perfect-information game is a set of actions for every node. The mixed strategy is a probability distribution over pure strategies. We can

define a support the same way as in a normal-form games. The support of mixed strategies in a perfect information game is a set of pure strategies.

The tree representation allows to use methods for pruning in the trees to find the pure strategies with the highest utility. We can use the minimax and the alpha-beta pruning to find these strategies.

Every finite perfect-information game can be transformed to a normal-form game. The problem is it can exponential grow up in a size. However, not all normal-form game can be transformed to a perfect information game. This ability to be easily transformed as a normal-form game can be used for finding dominant strategies or simply iteratively remove the dominated strategies and then the reduced game transform back into the extensive form.

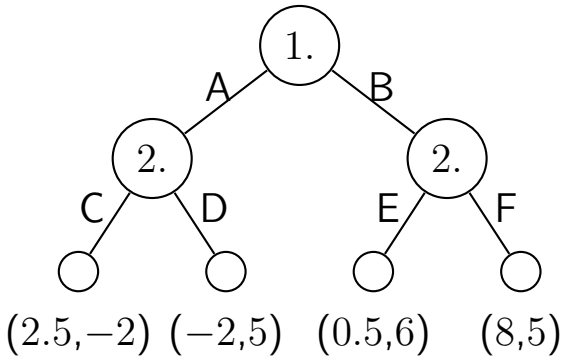

Figure 2.2: An extensive-form game

The Figure 2.2 is the two-player perfect-information game in the extensive form. On each node is written a number of the player, each edge has a letter, which is a name of an action. Under each leave is written a couple of a numbers. The first number is the utility for the first player and the second one is for the second player.

Now lets, look at the same game, but as a normal-form game. A simple extensive form is transformed into a bigger normal-form game. A normal-form game has 8 final states while an extensive has only 4. That means some of the states are duplicates.

	(C,E)	(C,F)	(D,E)	(D,F)
A	2.5,−2	2.5,−2	−2,5	−2,5
B	0.5,6	8,5	0.5,6	8,5

Figure 2.3: A game from Figure 2.2 in a normal-form game

2.3.2 The Games with the imperfect-information

Imperfect-information games in an extensive form allow us to represent a game with mutual ignorance of actions. An imperfect-information game is an extensive form game in which each players choice nodes are partitioned into information sets. If two choice nodes

are in same information set then an agent cannot distinguish between them. This means the result of a pure strategy of a player doesn't have the same result utility every time it is played.

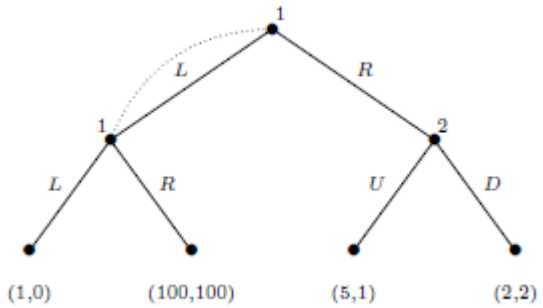

Figure 2.4: An imperfect game with an imperfect recall[14]

On the Figure 2.4 we can see an imperfect-information game. It looks like perfect information games, but two nodes are connected with a dotted line which represent an information set.

The information set is considered as a choice node, which can actually be a set of nodes. This means it doesn't reduce a total number of actions available to a player, but he will get different results for his actions. In the extensive form we can define a new type of a strategy called a behavioral strategy. "The behavioral strategies are randomizing independently at each information set rather than randomize over complete pure strategies. And so, whereas a mixed strategy is a distribution over vectors (each vector describing a pure strategy), a behavioral strategy is a vector of distributions."[14] A player using a behavioral strategy can randomize in each information set.

A perfect-recall game is a game where every player remembers his moves. Every perfect-information game is a perfect-recall game. The specialty of games with a perfect recall is that mixed and behavioral strategies are same. That means any behavioral strategy can be transformed into a mixed strategy and vice-versa.

Chapter 3

Monte Carlo Tree Search

3.1 Introduction

The Monte Carlo tree search (MCTS) is a heuristic search algorithm for finding optimal strategies in games.[3] The properties of the MCTS:

- algorithm is proven to converge to an optimal solution
- algorithm doesn't require to build a complete tree, it is building a tree during search
- some games have an exponential time for the convergence

The MCTS has small memory requirements and has a very fast rate of the convergence. It is used mostly for online problem solutions. There are multiple algorithms based on MCTS. Two most known are the Exponential-weigh algorithm for Exploration and Exploitation(EXP3) and the Upper Confidence Tree Search(UCT). In my study I focused onUCT.[3]

The optimal result is the Nash equilibria strategy profile. When a strategy is not optimal, we compute expected values of all players best responses for the strategy. From these values we compute the exploitability. We use the exploitability to compare progress of MCTS algorithms. If MCTS converge to an optimal solution, then the exploitability converges to zero. So if the exploitability is not converging to a zero, it means UCT has computed a wrong strategy profile.

The rate of the exploitability is considered as the MCTS converging rate. This means the rate of the exploitability getting near zero. UCT converges mostly with the $\sqrt{\log n}$ rate. The EXP3 converges with $\sqrt{n * \log(n)}$, where n is number of the simulations. That means in some games it is slower than UCT, but converge in all of them with constant speed. Difference in convergence rate is $\sqrt{n}$, which is bigger than the converging rate of UCT.[13]

3.2 Monte Carlo Tree Search

MCTS consists in employing principles recursively on all decision nodes in the game tree. Each round of MCTS consists of these four steps[5] :

- Selection: starting from the root, select successive child nodes down to a leaf node
- Expansion: unless the game ends, if a game doesn't contain a child node than create a child node
- Simulation: play an action from node, which one depends on the heuristic used
- Backpropagation: using the result of the playout, update information in the selectors on the path from leave to root

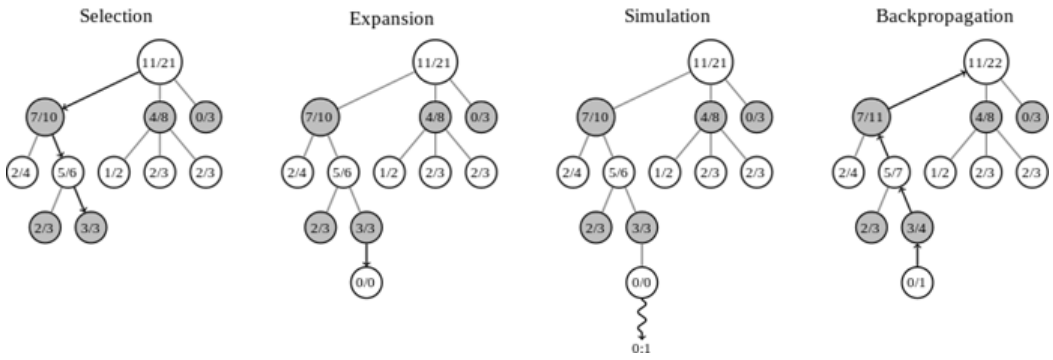

Figure 3.1: MCTS in each step[4]

The heurisitc is used mostly in the selection and the backpropagation step. The Expansion and the simulation are same in both implementations of MCTS we used.

3.3 UCT

UCT is MCTS using the Upper confidence Bound algorithm in each node[2][7]. The Upper Confidence Bound is the deterministic algorithm for solving a Multi-armed bandit problem. It is used to find the bandit with the biggest expected value in a probability distribution. UCT chooses always the maximum value of an action in the selection step, according to the value each action have. In the backpropagation we update an average value, a number of all actions played and a number of a played action [11]. Using this formula: $v + C * \sqrt{(log(n)/n_i)}$

- v is an average value of a descendants
- n is a number of all actions played
- n_i is a number of i-th action played
- C is a constant

The constant C is specific for every game.[1] It provides uniform distribution at the start of a game. UCT is the deterministic algorithm with no randomness. In the first iterations it behave as an uniform distribution, after that it starts to pick an actions with the higher

expected utility. To obtain the mixed strategy, we use the mean collector. Each action will have its probability as n_i/n.

In imperfect games UCT proves to find a strong strategy[11], but doesn't necessary means it finds the Nash equilibrium[8]. There are many ways to improve the convergence of the strategies to the optimal solution of UCT. We used the sliding window and the non-deterministic approach as UCT modifications.

3.3.1 Sliding windows

This modification use only the last t simulations for computing of a selection, where t is a constant for a duration of the algorithm run. It will increase the memory requirements for computing. Some state can become an unstable and the strategies will not converge to strategy profile, but will constantly change strategies.

The modification converges faster in the most cases than the original UCT. The original will need $\exp n$ simulations for changing a strategy. However a restriction to a constant number of simulations will not increase a number of simulations needed to learn a new strategy over a time. Result is a better chance of the convergence.

If we increase t we get that more simulations are remembered, better the precision, more stable solution, but bigger memory requirements and it takes more time for converging to an optimal solution. When we decrease t we get that less simulations are remembered, faster convergence, with smaller memory requirements, but with a less precision and a less stable solution.

3.3.2 Synchronization problem of UCT

A slightly modified game of the Rock, Paper and Scissor is a problematic game for a non-modified version of UCT. UCT will play a balanced strategy profile rather than converge to the Nash equilibrium. The balanced strategy profile utility gain for a both players is the same as a game value which is 0.5. It plays mostly 2nd action, because it seems to have a less dispersion than the other actions. If it tries to play an other action it will be getting a less utility, because UCT will start to play the best response to the other action. This means UCT selectors synchronized on playing certain pairs of actions.[12]

(0.5,0.5)	(0.75,0.25)	(0,1.0)
(0.25,0.75)	(0.5,0.5)	(0.55,0.45)
(1.0,0)	(0.45,0.55)	(0.5,0.5)

Figure 3.2: A modified RPS game

On Figure 3.1 we can see the modified game. The Nash equilibrium is to play all actions with uniform probability. Instead UCT decides to play a first player strategy as 3rd action 10.560%, 2nd action 80.719% and 1st action 8.721% and a second second player strategy as 2nd action 79.740%, 1st action 8.366% and 3rd action 11.894%. Resulting strategies have 0.1445 exploitability after $429,889,708$ simulations. From collected result we can see that it plays more one strategy, which seems. This result can be fixed by modifications.

3.3.3 Nondeterministic UCT

The idea of an algorithm is to add randomness to UCT. In a process of a selection we first count a number of values which are higher than the maximum minus an epsilon, than use an uniform distribution to choose an action to be simulated. This modification changes completely UCT form a deterministic to a non-deterministic algorithm. It allows better convergence of UCT.

The problem of classic UCT is that, if action values are near a game value it will never converge to the Nash equilibria. Example for this behaviour can be Rock, Paper and Scissors. It will choose a stable strategy profile near the Nash equilibria rather than Nash equilibria.

It has better memory requirements compared to the Sliding window. It converges in more cases than UCT. The only con is it need more time for selection. So overall it seems the best of them.

Chapter 4

Experimentation

4.1 Introduction

The first idea was to examine UCT and find its a speed and a problems of convergence. Lot of studies focused on great empirical results achieved by using UCT, but none about its convergence in imperfect-information games. Which are games problematic for UCT and what are the problems? Which classes of games converges faster than the others and why? Why in some games UCT converges faster than in another games? Which signs show a wrong behaviour of UCT, for example not converging to the Nash equilibria? Which modification is the best? How can we improve the results of UCT?

The first steps were to find games, in which UCT has unusual behaviour. Then we created a hypothesis, based on found games. We tested our hypothesis on multiple games, to see if the behaviour is stable or it's rare. During testing, we systematically changed constants and observed behaviour of UCT and its modifications. We repeated it for every modification.

We started first with simple matrix games. We transformed normal-form games into extensive-form for purpose of using them in testing of UCT. From a variety of games we chose two-player zero-sum games, because they are the easiest to observe problems. The matrix games allow us to observe the UCB problems and help us to create systematic methods of observation. We used those methods to examine more complicated games. We will test games on a both modifications to see the difference between them and directly compare them. We set the maximum number of simulations to a 430 million. After this simulations we didn't observe new phenomenons and more simulations will only prolong experiments to an unbearable time.

4.2 Implementation

We decided to use a game theory library from Czech Technical University as a framework because we get its code in a java and it is under the general public license. So we can observe the code, add our code and modify the code to our purpose. The biggest problem of this framework is that it isn't commented so it took a long time to learn how to use this framework. After some time of creating games, which was time consuming and didn't have a lot of successes we decided to use Gamut[10] for creating random games. Gamut is

a set of a game generators for testing game-theoretic algorithms. We didn't find Gamut, which supported extensive form games with an imperfect-information. So we decided to use a Standford gamut because it is easy to use and has a generator for a random two-player zero-sum game.

The CTU framework offers multiple domains of a game for the purpose of testing. We choose to modify the RPS domain, which is short for a Rock, Paper, and Scissor. We chose it for the reason it is a simultaneous two-player zero-sum game. One domain of a game consists of four classes. Those are the classes gamestate, expander, info, and action. We modified them for our purposes mostly adding an option for multiple round games. For an algorithm, we choose an ISMCTS algorithm, which is short for an information set Monte Carlo Tree Search [6]. The UCT selector has a one class, which was used for a testing modifications. The UCT selector had already implemented the non-deterministic modification of the UCT.

A harder part was to include a gamut into a program. We obtained gamut in *.jar format. We were unable to generate games via java code and decided to run it from a command line. So we implemented a function that calls a gamut and then we coded a class for reading the matrix from a file. We used a variety of games provided by the gamut.

In the game library already was implemented a solver of the Nash equilibrium for zero-sum games as an LP-program wrote for an IBM Cplex. It wasn't compatible with RPS domain because it had a problem with simultaneous games. So we created a class, which solve this problem by creating pure strategies from actions. We were unable to implement this solution for games with more rounds for each player. The exploitability counter was also implemented in the game library. The dominance solver wasn't implemented in the game library, so we implemented it using a Linear program in an IBM Cplex. We tested separately every strategy for being dominated by other strategies. The constraints are the utility values of each strategy except the tested one and the function is formed as utilities of a tested strategy. We added constrains, for a vector so he must contain only non-negative variables and must sum up to one.

We increases iteratively number of simulations in steps. The problem of epsilon is that on the 20th iteration it reaches the value of 485,165,195 simulations, which is the maximum amount of simulations we considered for testing. If we use epsilons powers we would get only less than 20 nodes, before it will be too many simulations needed to simulate the next step. We would need to use a formula of $e^(s/20)$, where s is the number of step. The other problem is that the first few iterations will be very small and hard to observe anything. So we create a formula $16 + 1.8^(s/8)$. We chose to have a base of a power 1.8 rather than e, for the reason that we got more data for an observation. Some data can be misleading as we didn't increase the number of simulations exponentialy. We saved all data in *.text files, which we than used for observations and creating graphs.

4.3 Iterative Dominance

In this experiment, we observed an impact of an iterative dominance on UCT. The game on the Table 4.1 has the 2nd and the 3rd column weekly dominated by the 4th column and the 1st row is also dominated. When we reduce the game we observe that the 1st column is dominated and after removal of the 1st column we see the 2nd and the 3rd rows domination.

2.0	3.5	3.0	2.0
2.1	4.0	1.0	1.0
3.0	1.0	1.0	1.0
1.0	2.0	3.0	2.0

Table 4.1: A normal-form game

2.0	2.0

Table 4.2: A reduced game

The game value is 2. The Nash equilibrium for the first player is to play the 1st action and for the second player to play the 4th action. It has no problem converging to the Nash equilibrium.

UCT has no problem with a dominance. The dominated actions probability of being played was decreasing rapidly. All of the dominated actions have less than 0.0001 probability of being played, after 3 million. From this experiment, we got an idea that dominance is less relevant than the utility values. Dominated actions have the utility values smaller than the rest of actions, that means the UCT will always stop playing them fast. The iteratively dominated actions have near two times higher percentage of being played than the dominated ones. They converge with relatively same speed to the zero. We don't consider a dominance to have a major issue in a convergence of UCT. We think that more important than dominance is the utility values gained from the actions.

4.4 Matrix games

We generate multiple games and then take the one with problems to converge to the Nash equilibrium. On this one, we choose to change a constant and see if it will converge. We wrote to a text file strategies and a selector values to observe UCT. In the Figure below we can see a game in a normal form, which has a problem of a convergence. The Nash equilibrium strategy for the 1st player is to play the 2nd action in 40% of the time and the 1st action 60% of the time. The second player Nash equilibrium strategy is to play the 2nd action in 70% of a time and the 1st action 30% of a time. The game value is -1.35.

-4.0	0.0
3.0	-3.0
0.0	-2.0

Figure 4.1: Game in normal-form

4.4.1 UCT

We tested classic UCT with no modification. UCT had a big problem with this game. It chooses to play the dominated strategy instead of playing Nash equilibria. The first player

strategy is to play the 3rd action with 57.483%, the 2nd action with 0.015% and the 1st action with 42.502%. The second player strategy is to play the 2nd action with 57.508% and the 1st action with 42.492%. We can clearly observe that classic UCT will not converge to the Nash equilibria. This is caused by the synchronization when a first player played second action the second player played the second action. This decreased an action value of the second action so it is not played enough to change it action value.

4.4.2 Non-deterministic

We tested UCT with the non-deterministic modification using epsilon 0.0001. After 429,889,708 simulations, the first player strategy is to play the 3rd action with 67.8% probability and the 1st action with 32.2% probability. The second player strategy is to play the 2nd action 67.7% of a time and the 1st action 32.3% of a time. The exploitability is 0.166. We can clearly see it's not converging towards the Nash equilibria. The expected utility for the 1. player strategies is -1.288 for the first action, -1.065 for second action and -1.354 for third action.

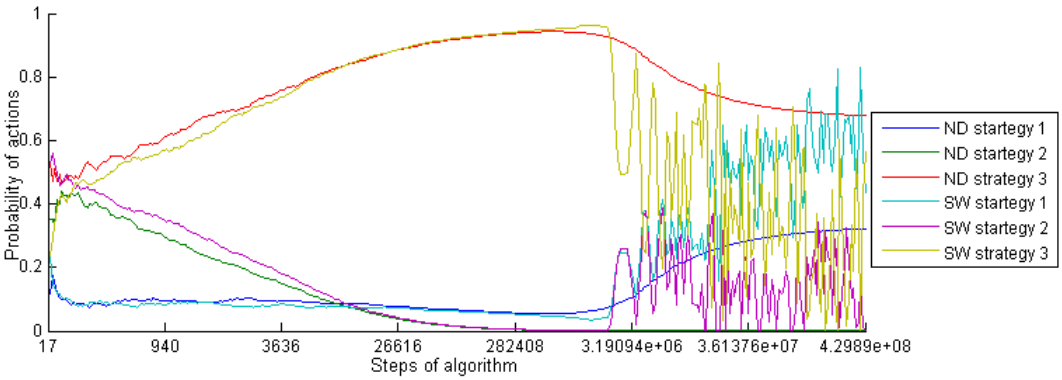

Figure 4.2: First player strategy

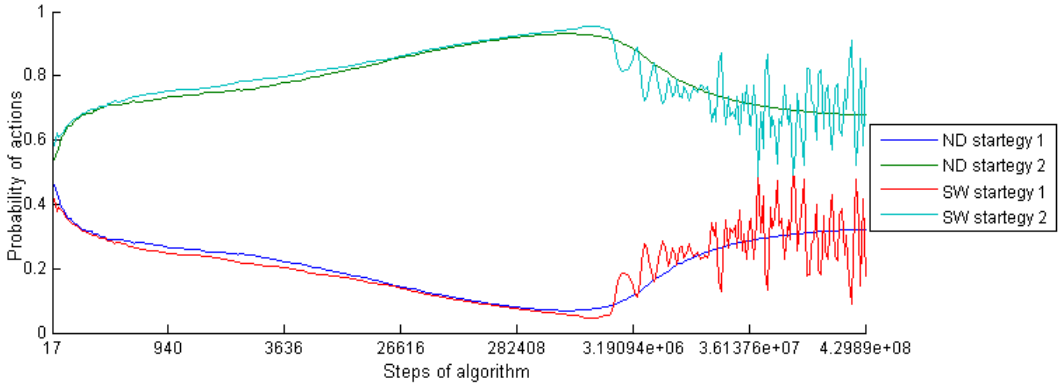

Figure 4.3: Second player strategy

Despite the fact that second action has the highest utility value, it is not played for the fact that UCT has it value much lower. The interesting fact is that the 3rd action is dominated

by the mix of 1st and 2nd actions. We can observe a problem that an exploitability reached its minimum and will only slightly decrease. The algorithm finds a local minimum for an exploitability and has a problem to reach the global minimum.

The Figures 4.2 and 4.3 show the first and the second player strategies as they evolve during simulations. We can see at the start the non-deterministic plays the 2nd action, but the 2nd action is slowly stopping of being played. The second player strategy doesn't change much through simulations. So the problem is in an initial strategy of the 2nd player. The interesting fact is that near 1,200,000 simulations it starts to converge to the local maximum. Even after further iterations it will not change to the Nash equilibrium.

The curve of convergence look-alike logarithmic. On the Figures 4.4 and 4.5 we can see a change of action values of UCT selectors over a time. This value is used to choose an action, which is going to be played. We can see the actions being played converges to a game value, also the difference between the action values is getting smaller. We observe the same behaviour in values as the second player strategy change near 1,000,000 simulations, the values of the Non-deterministic changes as well.

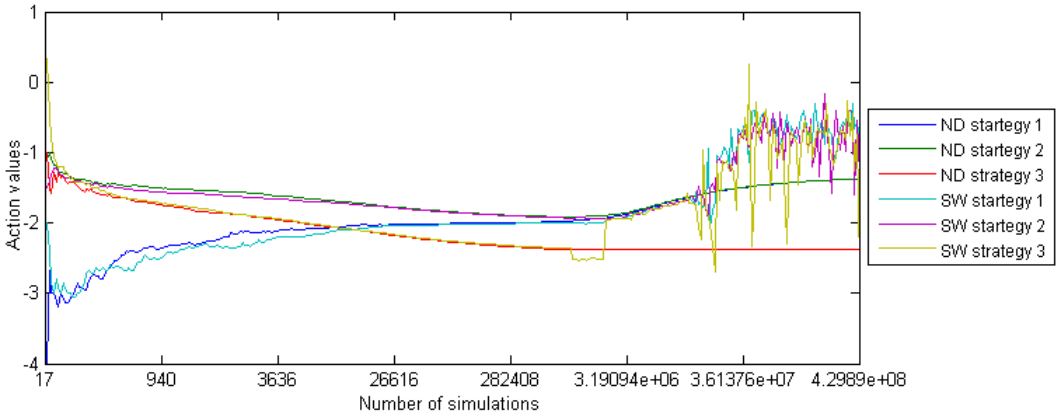

Figure 4.4: First player values

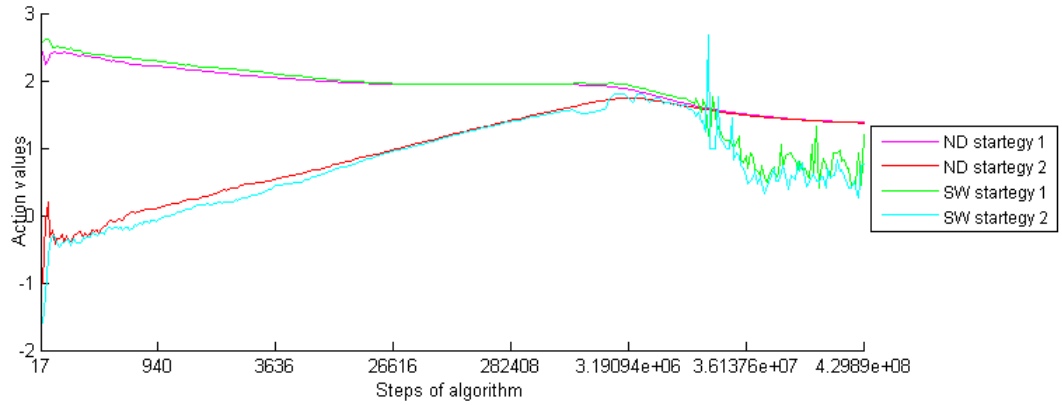

Figure 4.5: Second player values

The change of values will happen sooner as a change of a strategy. That means we can predict a change of a strategy, based only on a changing value of a selector. We know the

non-deterministic modification of UCT needs an exponentially more simulations to get a new strategy. If we could detect a change in UCT selector values we could be able to improve results of the nondeterministic modification of UCT.

The non-deterministic modification of UCT in this game got stuck in a local maximum. The action with the biggest probability for the first player isn't even in support of the Nash equilibrium. That means if the other player played the Nash equilibrium, the first player will not gain the value of the game, but instead something smaller. The problem is it will take a very long time, to realize that the second action actual gain more utility than the first or the third action. So the problem is that at the start the action values were small, which results in small number of simulations after the start. Later it was not played enough to change its action value and the $log(n)/n_i$ increases with a very slow rate.

4.4.3 Sliding window

The sliding window modification of UCT has the window of a 1,000,000 simulations. The Nash equilibrium and number of simulations is the same as in the non-deterministic modification of UCT. The resulting strategy is to play the 3rd action with 8.45%, the 2nd action with 24.63% and the 1st action 66.92% probability of being played. The exploitability is 0.468.

The exploitability is higher than for the non-deterministic modification of UCT, but the first player plays strategy nearer the Nash equilibria than the non-deterministic modification of UCT. We observe multiple radical changes in strategy, this phenomenon is caused by forgetting simulations and keeping the only limited number of simulations to be used as a base for a selector. Till it reaches a million simulations it uses strategy, which is similar to the non-deterministic modification of UCT. After reaching it a full size, it starts to behave a more random with the fast changing strategy. The random behaviour is caused by the increasing number of simulations to the next observation while the amount of simulations needed to completely change a strategy doesn't increase as in classic UCT. This fact makes Sliding Window look as an algorithm which very fast change counted strategy profile compared to the nondeterministic modification of UCT. The rate of changing is it the biggest plus and a minus at the same time.

It is possible that during the experiment sliding window modification of UCT has created better strategy, than the sliding window modification of UCT. The only problem is that if it finds a good strategy profile it will keep changing and will not stay in a strategy profile. So the strategies in SW doesn't converge as they do in the sliding window modification of UCT. This means that the resulting strategy will not get better over a time rather it will constantly change. It creates a problem of choosing the right number of simulations, because a small number of iterations can be better than a higher number of simulations.

4.4.4 Problem of a game

UCT is an algorithm to find strategies of games. If a game has the Nash equilibria, UCT is used for finding one of the Nash equilibrium strategy profiles. So a basic idea is that it will play all actions from the Nash equilibria support or at least the most probable action will be from the Nash equilibria. So if we will play the game and enemy plays the Nash equilibrium

strategy we still get the game value. This game proves that those ideas are wrong. That means there exist a game in which algorithm choose to play a strategy which has a higher probability than any action from the Nash equilibria support. In this game, the action is dominated by the combinations of the rest of actions.

4.5 Test of a large number of games

After obtaining a method for testing games, we decided to run a complex test. For the test, we thought of testing all two-player zero-sum games with a 3 actions for an each player. We limited a utility to -1, 1 and 0. The idea is to find all non-converting games and find reasons why they don't converge. We chose these games because they are very simple, we hoped for a result which will help us to better understand UCT. The total amount of 19680 games were tested. Initially, we let each game to be played 3,190,936 simulations. We used the non-deterministic modification of theUCT with the 0.0001 epsilon. We then took all games with the exploitability more than 0.25 and run those for the 14,130,408 iterations. We then choose from the results games with the exploitability higher than 0.25 and run those games for $99,732,004$. After that, we didn't find a game with higher than 0.25 exploitability. The maximum after the last test was 0.146 exploitability.

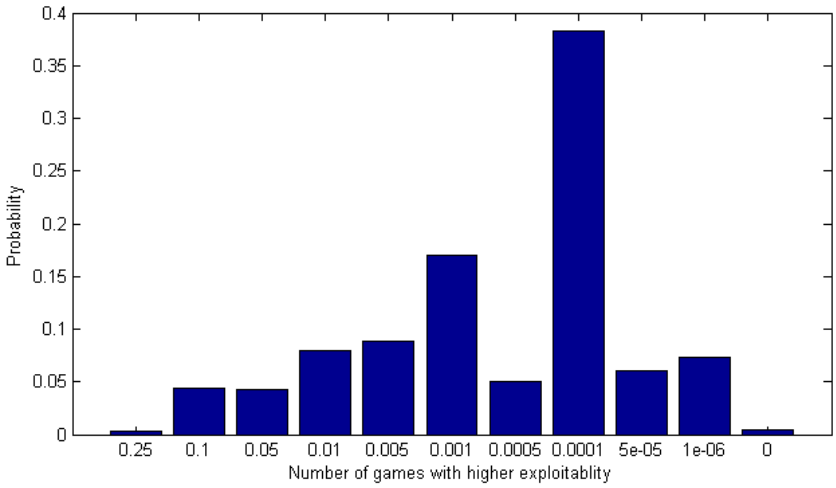

Figure 4.6: Distribution of exploitability

On the Figure 4.6 we can see a distribution of an exploitability of a first test of games. This graph can be misleading because there are all 19,680 games involved, without reducing the similar games. We didn't find a generator of matrix games so we tried all games. Different games have different number of possible variations. This graph can show the overall distribution of an exploitability over all games. The most games will have after 3,190,000 simulations an exploitability between 0.25 and 0.000001. None games have an exploitability higher than 0.5. We can see that most games converge fast to the Nash equilibria. From distribution we observe that only 10% of games have higher exploitability than 0.05, which we consider as games with problems of convergence. Around 38% of games are between 0.0005 and 0.0001. In this largest groups are games with a pure strategy in the Nash equilibria.

4.5.1 Changes in counted strategies

We got an idea to look on how often UCT changes the counted strategy direction during simulations. We used a stored data from the last experiment. The main reason for this experiment is to find the percentage of games which change the direction of a strategy convergence. We thought that there are points during which UCT has a higher probability of changing a strategy. Also, we observed the optimal length for testing if the directions of a strategy convergence have changed. We counted a percentage of strategies, which changed the convergence direction of their actions. The change of a direction means that if in the last step an action increased their probability and in the new step the action decreases the probability then we called it a change of the direction of a convergence. We used all tested games and extracted from them the overall percentage of the directions of a convergence, which has changed. On the Figure 4.7 we see a graph of a percentage of games, which changed the direction of a convergence compared to the last observation of actions over simulations.

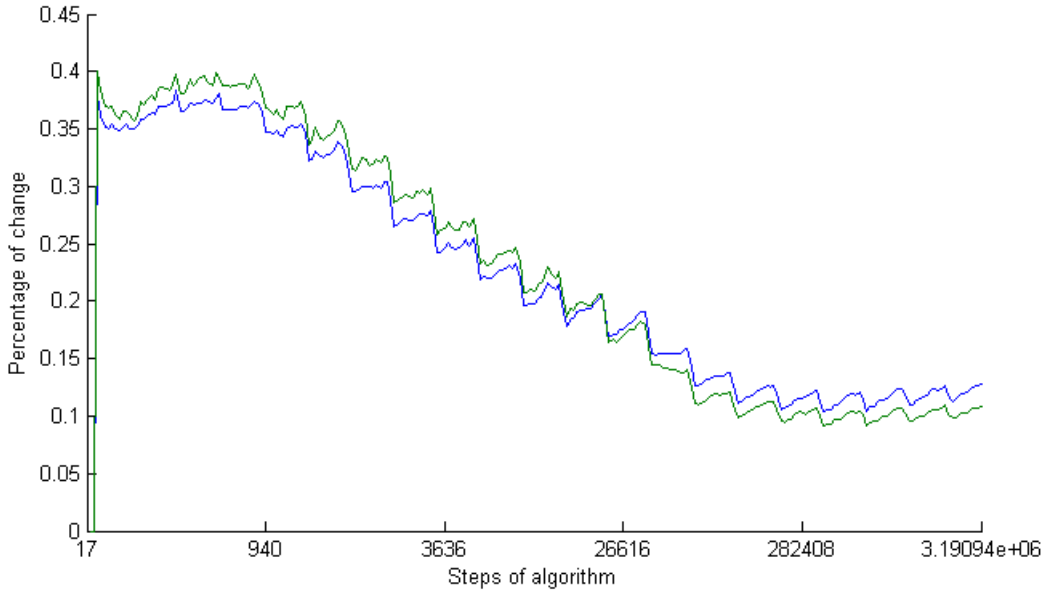

Figure 4.7: A graph of UCT changing counted strategies

On the graphs we see a zigzag pattern of a line, this is caused by a small diversity of games, the range values are small and fact that it's an average value of large number of games. We believe that most of this bumps weren't created by changing strategy, but the fact that strategies oscillated near a converging strategy and we look at a point in which it slightly changed its strategy. Also the fact that we increased number of iterations in cycles, that means we increase every 8th step number of iterations by multiplying it by 1.8 4.2. To know if the pattern is relevant or it is just coincidence we decided to do more tests and compare different epsilons. We desire to know if the pattern remains on randomly generated games.

We see that changing of the directions of a convergence get less frequent around a million simulations and then it little increase. The biggest changing percentage is on the beginning,

which is 40% and the lowest is near a 300,000 simulations, which is about 12%. The percentage is decreasing till 300,000 simulations, then it stopped decreasing and acts as constant. The unexpected result is that in the end it slightly increased a percentage of changes. We think that the step size overcome the critical point for a changing strategies, which caused the percentage increase of strategy changes. This shows that in every step has at least 12% probability of changing strategy than the last one. So the directions of a convergence change a lot during simulations.

4.5.2 New test of changes

We created 2000 random games and run each about 14 million simulations. We use the same steps system as before. We run it twice with two different epsilons. One epsilon is 0.1 and the other is 0.0001.The last point is 14,130,108 simulations. We show with this experiment a confirmation of some facts and showing the difference in epsilon. The percentage decrease normally again to 1 million and then it starts to oscillate. The increase about 3 million has not happened as before. The percentage starts at 35% and goes down to 5%.

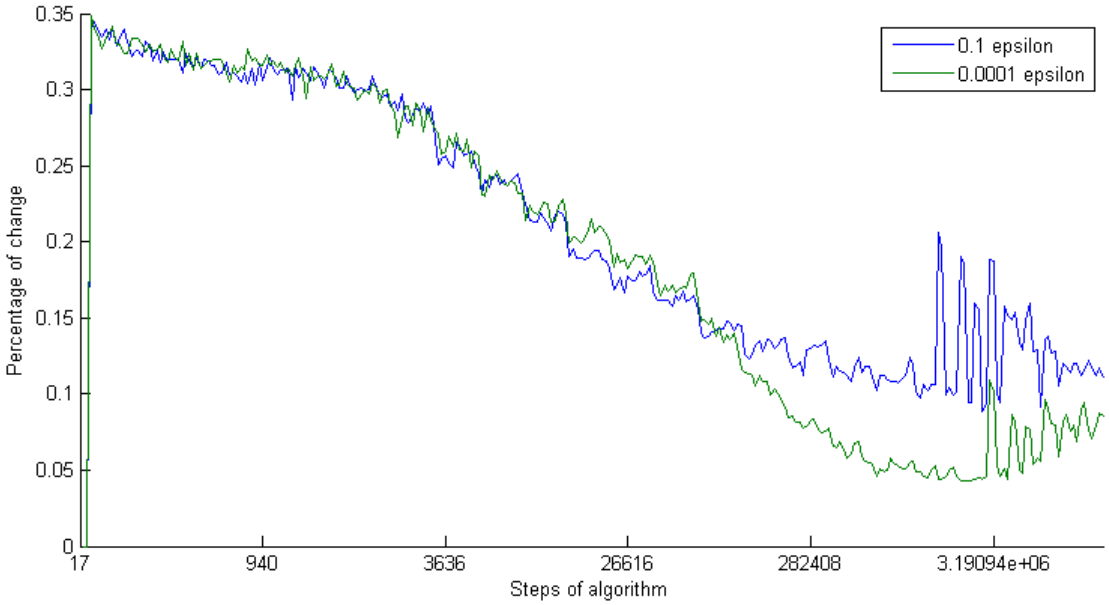

Figure 4.8: A comparison of two epsilons

On the Figure 4.8 we see a 0.1 epsilon and a 0.0001 epsilon comparison. We still can see an oscillation near strategies, but is much smoother than the generated games. This indicates that it was caused by similarities in the games. The difference starts to show off about 100,000 simulations the 0.0001 epsilon changes less often than the 0.1 epsilon. It is caused by the bigger window for the random changes. The 0.1 epsilon hasn't got a big decrease and then increase as the 0.0001 epsilon had. So the bigger epsilon means it changes the strategies more often than the lower one which is a logical thing to happen. The big deflections are mainly caused by a point were action values difference are smaller than epsilon. We assumed this by the fact that both epsilons have big deflections, but at different simulations and different amplitudes.

4.5.3 Problematic game

Now let's look at interesting games we find during a testing of a generated games. A game was run with a 429, 889, 708 simulations. On the Figure 4.9 we can see a game. A game value is 0.2. The Nash equilibrium strategy for the first player is to play the 2nd action 40%, the 1st action 40% and the 3rd action 20%. The Nash equilibria strategy for the second player is to play the 2nd action 40%, the 1st action 40% and the 3rd action 20%. This game is the member of a larger domain. Every game which has exactly one -1 value in each row and each column, has exactly one 0 and other places are filled with 1 have the same behaviour as this game.

0.0	1.0	-1.0
1.0	-1.0	1.0
-1.0	1.0	1.0

Figure 4.9: A normal-form game

An exploitability is 0.062, which seems not much. A counted strategy for the first player is to play the 3rd action 16.7414%, the 2nd action 38.9712% and the 1st action 44.28742%. A counted strategy for the second player is to play the 2nd action 38.9666%, the 1st action 45.9178% and the 3rd action 15.1156%. We can observe that probabilities aren't far from the Nash equilibrium. So it might seem that this game is going to converge to the Nash equilibrium. It will converge very slowly compared to the other games we tested, but it converges towards the Nash equilibrium.

The exploitability seems to be small, but it is big when we consider maximum range is 2 and UCT used nearly half of billion simulations and wasn't change much from the start. UCT has a problem with a synchronization. This is a similar problem we can see that UCT plays strategy profile near the Nash equilibrium, but it converges very slowly to it. The resulting strategy is not the Nash equilibrium, despite the fact we use the non-deterministic approach with randomness.

4.6 Simultaneous games

We decided to observe imperfect-information games. The first idea was to create a small two player games, for an easier observance of a problematic behaviour. We continue to keep it simple so it will be a zero-sum game. The problem of the information set is that an agent doesn't recognize in which node he exactly is. UCT has problems to correctly choose a best average value when there are two or more independent random distributions. We decided to have a simultaneous game because it's the very simple type of an imperfect-information game with perfect recall. We didn't consider a good idea to use chance nodes as a part of a game because it will be problematic to determine if the problem is in the set or in chance node distribution.

4.6.1 First problematic game

On Figure 4.10 we can see a game tree of a very interesting game. Both players have a 4 possible actions divided into 2 information sets. Each player have a 3 selectors. We will compare the modifications of UCT on this game. A game value of a game is 1.1. The Nash equilibria strategy for the 1st player is the (A, F1) action with 35% and the (B, E2) action with 65% probability. The Nash equilibria strategy for a 2nd player is to play the (C, G1) action with 70% and the (D, H2) action with 30% probability.

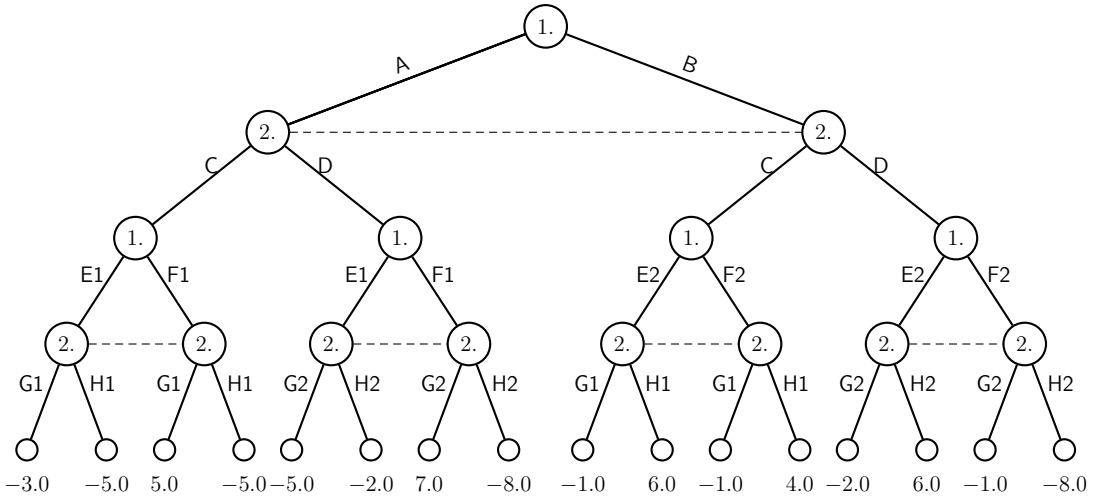

Figure 4.10: A game with an imperfect information

An epsilon for the non-deterministic modification of UCT is 0.0001, 0.01 and 0.1. The epsilon 0.0001 is good for a testing improvements because UCT is just very bad in choosing reasonable strategy on it. The idea of testing a near epsilon is to show that some games have an epsilon where they converge and even making it smaller by a decade can change their behaviour. Proving this means it is necessary to change epsilon for a specific games. On the Figures 4.12 and 4.13 we can see a graph of an action probabilities over the simulations and on a Figure 4.11 we can see a graph of an exploitability over the simulations. On a graph of strategies, we can see only 0.1 and 0.01 because it will be very confusing to observe the difference between three epsilons.

Let's have a first look on exploitability. On the first look, it seems strange at some points. The most strange thing is that exploitability for 0.0001 epsilon grows. This strange thing is a big problem for UCT and we will get back to it later in improvements. So we will now compare directly the difference between 0.1 and 0.01 epsilon. First thousand simulations are almost same, because of a constant C of UCT. This constant change the algorithm to play uniform distribution over actions. This means that both epsilons are affected by this constant and cause of it play unifrom distribution over all actions. We observe that UCT with 0.01 epsilon has the same exploitability during the whole experiment. It will not decrease or increase during simulations, which proves that UCT doesn't necessary needs to change the exploitability. We can see that the exploitability of UCT with a 0.1 epsilon is near zero around 120.000 simulations and it will stop at the 0.0375 exploitability.

Let's now observe a strategy of players computed by UCT with the two different epsilons. The result strategy for the 0.1 epsilon is for the first player to play the (B, E2) action with

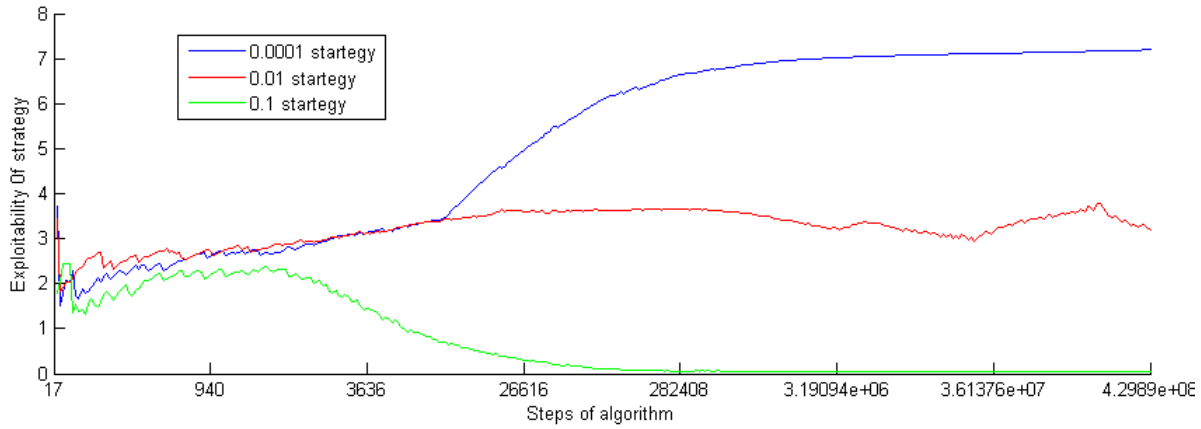

Figure 4.11: Distribution of exploitability

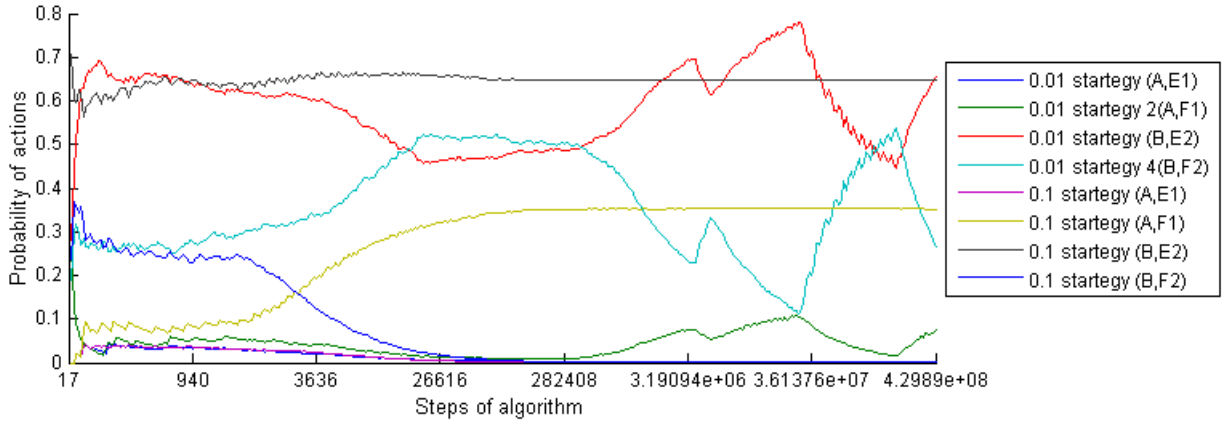

Figure 4.12: Strategy of first player

35.228% and the (A, F1) action with 64.772% for probability and for second player is to play the (C,G1) action with 49.979% probability, the (D,G2) action with 33.021% and the (D,H2) action with 16.978% probability. The result strategy for 0.01 epsilon is for the first player to play the (B, E2)action with 7.634%, the (A, E1) action with 65.588% and the (A, F1) action with 26.776 % probability and for the second player is to play the (D, G2) action with 88.811% and the (D, H2) action with 11.182% probability.

The noticeable difference between the two epsilon is that 0.1 epsilon is fast converging to a distribution while 0.01 epsilon had changed the strategy more than once. We will be observing the epsilon 0.01 for the moment The biggest problem is that it doesn't converge to the Nash equilibrium and it plays actions which aren't in support of the Nash equilibrium. The big mistake is the second player strategy, where the actions being most played is not in the Nash equilibrium support. This mistake is caused by the first selector which plays only the second set of the strategies. It would take a lot of simulations to change a strategy to play the Nash equilibrium.

The epsilon 0.1 compute wrong second player strategy. It is caused by a several factors. If the first player plays the Nash equilibrium the second player actions values are for the (C,

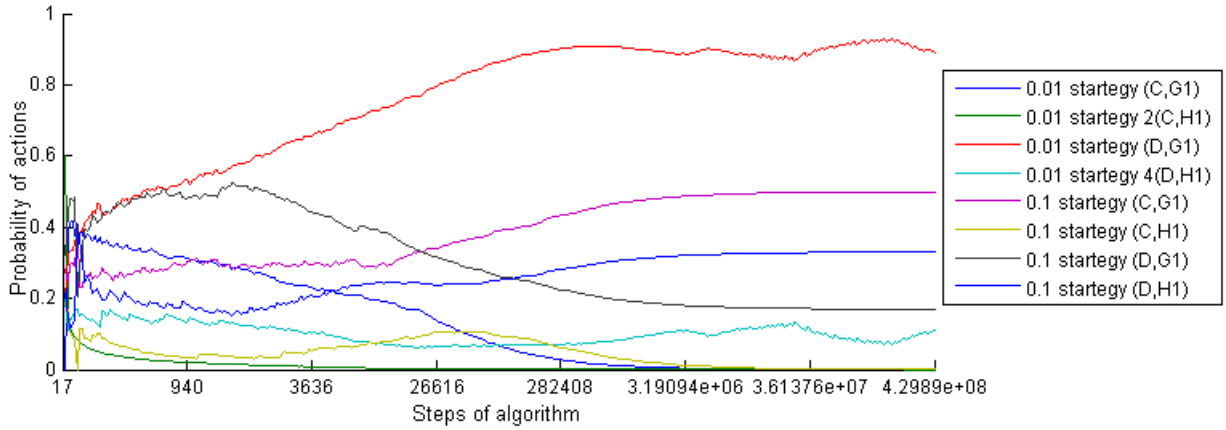

Figure 4.13: Strategy of second player

G1) action is 1.1, for the (C, H1) action is 2.15, for the (D, G2) action is 1.15 and for the (D, H2) action is 1.1. So the (D, G2) action has a value near the game value. The resulting utility values for the second player are for the (A, E1) action is -3.01, for the (A, F1) action is 1.05, for the (B, E2) action is 1.14 and for the (B, F2) action is -3.31. So the minimal distance between the Nash equilibrium action and the lowest non-equilibrium action is 0.05 for the second player and is 0.09 for the first player counted strategy which are both under epsilon. This cause uniform distribution over the (D, G2) and the (D, H2) actions. The exploitability for a 0.05 epsilon is 0.0136.

We showed with this experiment that some games require a very specific epsilon to converge to the Nash equilibrium. We tried the two near epsilons, but none of them actually converged to the Nash equilibrium. We showed that setting too high of epsilon can mislead. This means that exists games in which the non-deterministic modification of UCT algorithm wouldn't work. The problem is that for the high utility values we need high epsilon to make them play more action within the epsilon, but the limitation is the nearest utility value of an action for a player while other plays the Nash equilibrium strategy. It will be very hard to choose the right epsilon for games with high utility values or a big range of utilities. UCT used only with the constant epsilon will find games which the epsilon will be too big or too small to find the Nash equilibrium. Also trying different epsilon is an option, but it will require more time which is crucial in online computing of strategies.

4.6.2 Second problematic game

This game we tested on the non-Deterministic modification of UCT with the epsilon of 0.0001. The game value is 3.143. The Nash equilibrium strategy for the 1st player is to play the (A, E1) action with 7.143% and the (A, F1) action with 92.857%. The Nash equilibrium strategy for the 2nd player (D, G2) action with 85.714% and the (D, H2) action with 14.286%. When the first player will only play the (A, F1) action the minimum utility is 3, which is near the game value. The classic UCT will not play the Nash equilibrium, neither the non-deterministic version will play it.

The result first player strategy for the non-deterministic modification of UCT with a

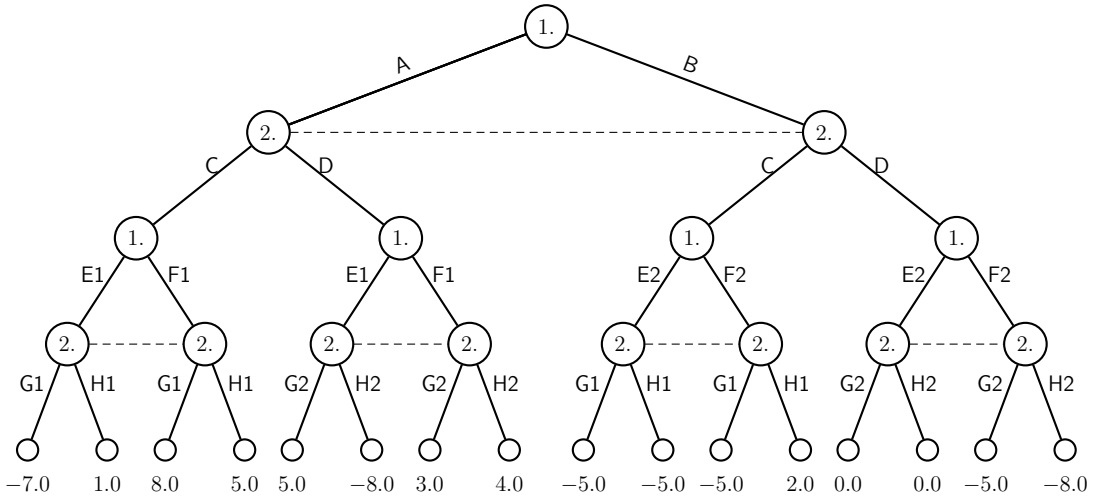

Figure 4.14: A game with an imperfect information

0.0001 epsilon is to play the (A, E1) action with 99.999% probability. The resulting strategy for the 2nd player is to play the (D, G2) action with 99.999% probability. The exploitability is the 0.996 because the best-respond to the (D, G2) action is actually the (A, E2) action with the resulting utility of 5. So the one player always reach near the Nash equilibrium, but the other will be far from it. We tested the different epsilon with the same results. We observed that for certain games UCT will probably never reach the Nash equilibrium for both strategies.

On the Figure 4.15 we can see a graph of the exploitability. The strange thing is that the exploitability starts to increase from the point around 200,000 simulations. This is caused by the point were action the (D, H) has the Nash equilibrium probability of being played. The action (D, H) will continue to decrease the probability of being played to zero, which will increase the exploitability. The non-deterministic modification of UCT should make enough randomness prevent these actions, but it fails to do so.

This game was created randomly. Look at the result at the (B, E) action and the (E, G) which is 0. We increase the utility value of the (B,E) action and the (E,G) action up to 200 for the epsilon 0.1, the exploitability for the computed game is 67.807 which is too big for a game with 3.98 game value. This modification increases the exploitability to whole new level. This shows that if there is the Nash equilibrium strategy which have a probability of actions near 1, UCT will have problems to reach the Nash equilibrium for the other player.

4.6.3 Third game

In the previous charts, we only looked at the non-deterministic modification of UCT now let's look at the sliding window modification. The memory requirements increase rapidly with each new selector, which makes the sliding window very bad in terms of requirements to run the modification. On the Figure 4.16 we see a game tree of a simple simultaneous game. The game value is 0.96. The Nash equilibrium strategy for the 1st player is to play the (A,E1) action with 39%, the (A,F1) action with 21% and the (B,F2) action with 40%. The Nash equilibrium strategy for the 2nd player is to play the (C,H1) action with 56%, the

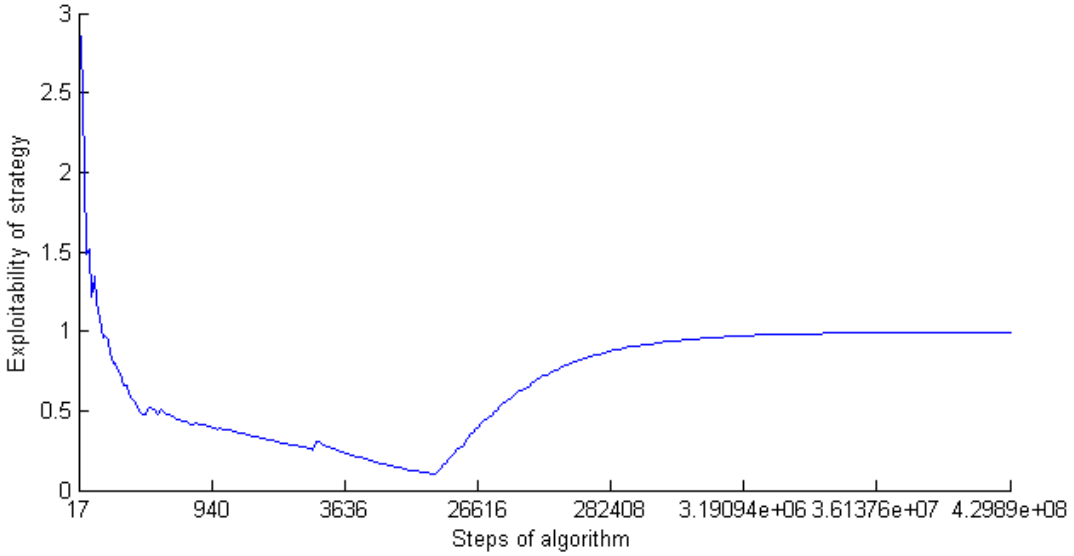

Figure 4.15: Exploitability of the game

(D,G2) action with 20% and the (D,H2) action with 24%. So the Nash equilibrium support contains almost all actions.

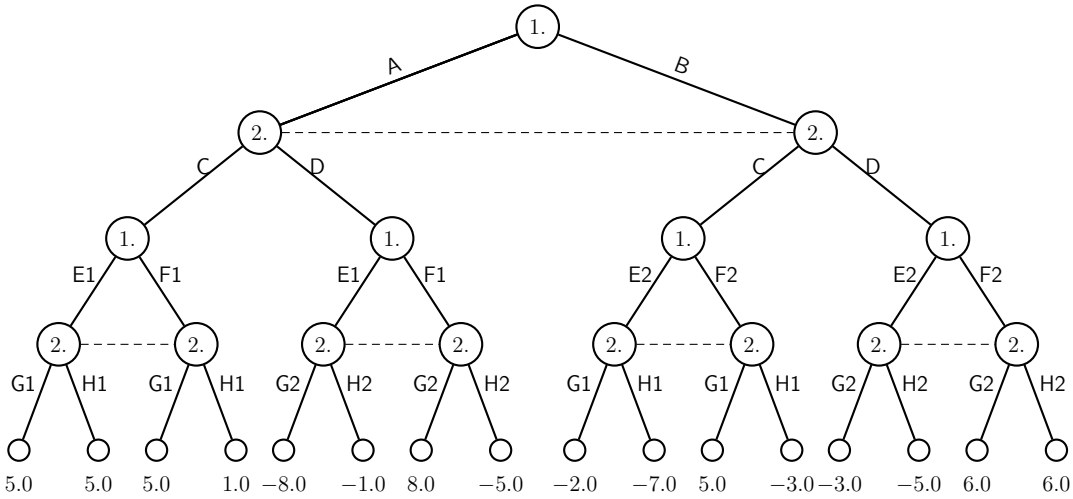

Figure 4.16: A game with an imperfect information

We use the usual 429, 889, 708 simulations. The sliding window was set to 70,000. The exploitability 1.463. The counted Strategy for the first player is to play the (A, E1) action with 29.452%, the (A, F1) action with 0.270% and the (B, F2) action 70.231%. The counted Strategy for the 2nd player is to play the (C, G1) action with 0.822%, the (C, H1) action with 41.104%, the (D, G2) action with 0.226% and the (D, H2) action with 57.847%. The first noticeable problem started with memory requirements every node needed their own queue, which dramatically reduce the size of the window because it needed to have 3 selectors for 4 possible actions. The smaller window makes it harder to find a stable strategy profile and so finding of the Nash equilibrium is harder. We will have the graphs only to 3,1 million

simulations cause behind this point is hard to see any pattern of strategy.

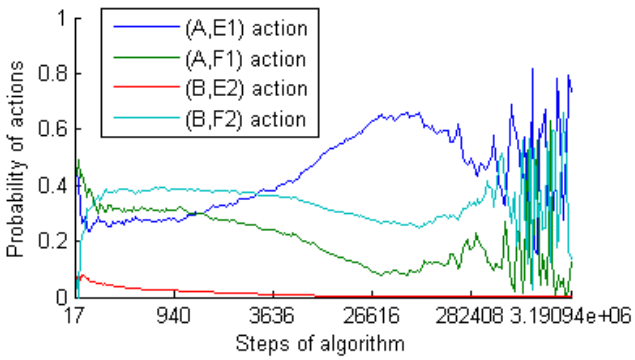

Figure 4.17: A graph of first player changing strategies

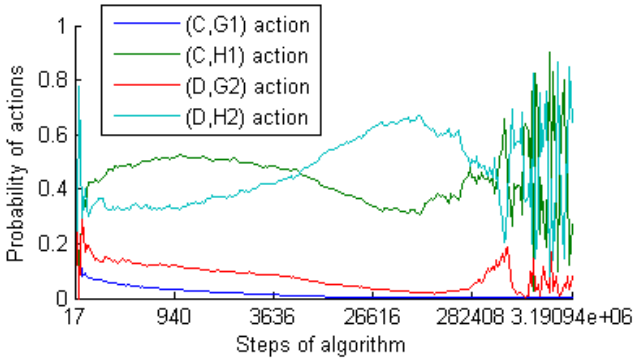

Figure 4.18: A graph of second player changing strategies

On the Figures 4.18 and 4.17 we see the sliding window counted strategies for both players. The (B, E2) action and the (C, G1) action are stopped being played at the start. These actions are not in the support of the Nash equilibrium strategy profile. So the Sliding Window plays only the Nash equilibria support actions. Also, the order of action probabilities for the first player is right, but the second player plays more the (D, H2) action than the (C, H1) which is not right. The best exploitability was 0.48 at 2600, then it rises up to 3.14 at 80,000 simulations and then it decreases slightly. Then it starts to change more rapidly cause of a small window the best value was 0.27 and the worst value was 5.5. The interesting fact is that it starts to play first player strategy near the Nash equilibrium, but the second player keep to change in the wrong order of action probabilities. This problem is caused by near values of actions and a wrong probability distribution of the actions of a first player. It suffers the same problem as the non-deterministic modification of UCT in the second game.

The small size of a window allows it to change a strategy without the need of an increased number of strategies. The one problem is that multiple selectors need to remember the value of an action, because the multiple selectors have it as different actions. This increase the memory requirements by a significant number, but also it makes a window for each selector and not the window for the whole UCT. It suffers from the same problem with stability as the simple matrix game did. The biggest problem is still that with increasing number of simulations it doesn't converge to some result. It's very possible that the strategy changes in cycle. But without an ability to count the exploitability we can't determine if the counted strategy is near or distant from the Nash equilibrium.

Chapter 5

Discussion and future work suggestions

The UCT has performed very well in perfect information games like a Go. The UCB algorithm has a problem with multiple random generators. The main problem of UCT is that it is a deterministic algorithm. We tried a two modifications, which make UCT handle a randomness. Both of them have some problems but are working in their own way. The first idea of improving was to detect the wrong strategy or the wrong result. In this case, we would use another algorithm, for example, the EXP-3 [13]. We observed if the strategy changes often it has the high probability of not being the Nash equilibrium. We were unable to distinguish the wrong from right strategies when UCT was converging.

We discovered that we don't need to look for number of actions, but rather to observe the changes of action values. When UCT converges to the Nash equilibrium the action values in a Nash equilibrium strategy get closer to each other and get closer to a game value. This means we can easily observe changing strategies, just by checking an action value. This allows us to just change selector for determining if the strategy changes.

The most problematic part is that UCT starts with the uniform distribution which is held for a first few thousand simulations. Then the $sqrt(n_i/log(n))$ will become very low. This allows actions with low value to be played at the start, which is good and bad in the same time. When the square root is near zero UCT algorithm will try to find the best strategy profile without the chance for actions with low action values to be played. We didn't find a method to improve this. We tried to change logarithm to third root, but it didn't work.

5.1 Non-deterministic

One of a bigger problem of UCT is that it needs increasingly more simulations for every new change of a strategy. We observed in tests that UCT is changing strategies a lot. It means it will constantly need more simulations for new changes. That prolongs the simulations needed to find the Nash equilibria.

If UCT finds a strategy profile, which is near the Nash equilibria it will converge to it. We know that changing strategy suggest that strategy is not in the Nash equilibrium. If we allow UCT to change strategy without increasing simulations, we will increase a chance of UCT finding the Nash equilibria. The idea is to somehow make UCT change strategies

without increasing simulations needed for a change of a counted strategy. This idea is behind the Sliding window, but it has some flaws.

We chose a different approach, instead of a limitation we chose to reset the selectors. This idea is based on the observation we made in the matrix game. We observed that we know the change of a strategy before it will happen from the action values used by UCT selectors. We write about it in Section 4.4.2 We choose to reset only counts of actions and don't change the action values. We picked 100 actions played for the reason that bad strategies will not be played a lot and allow played strategies to change the probability between them. We didn't want to change an action value because it will cycle. The number of simulations after, which we test a UCT selector for a change of a strategy is iteratively increased using a multiplier. During experimentations, we observed a percentage of games changing their directions of convergence.

```
if n = tn then
    tn= tn*1.5
    for i ≤ number of actions do
        if first test then
            store the first values
        else
            if direction is not stored then
                store direction
            else if action changed the way it converge before then
                set all counts of actions to 100
                set a number of actions
                stop for-cycle
            end if
        end if
    end for
end if
select an action
```

Figure 5.1: Improvement of the non-deterministic modification of UCT

This allows UCT to change a strategy more effectively. It creates a new problem because it can cycle. This issue can be easily, overcome by changing initial point or multiplier. The first game of simultaneous games in Section 4.6.1 for epsilon 0.0001 has ended with 7.192 with our improvement we were able to get 0.056. The matrix game had a problem with converging, but the improvement achieved 0.1013 exploitability. The UCT using our improvement counted the first player strategy as to play the 3rd action with 1.809%, the 2nd action with 40.330% and the 1st action 57.861 and the second player strategy is to play the 2nd action with 69.079% and the 1st action with 30.921%. We clearly demonstrated that in some games it is better than the nondeterministic modification of UCT and in worst case it has the same results. The non-deterministic modifications of UCT have a problem with the right size of an epsilon, which this improvement solves. This improvement has best results with games in which UCT fails to find the Nash equilibrium strategy profile, but it can also help to decrease the number of simulations needed to find the Nash equilibrium.

5.2 Sliding window

Overall the sliding window is a good modification for small number of simulations, it will not become much better with large numbers of them. The exact number where to stop algorithm is specific for each game so it is not an option to find it. It is impossible to always count exploitability and with its help determine the best strategy.

We got an idea to check the number and the distance of action values in selectors and increase the size of the window when it will converge. We got this idea from observations in Section 4.6 The main problem of implementing is how much to increase and, what is more, significant the number of actions or how close to each other they are. This still doesn't guarantee the best strategy, but increase the chance of counting and staying in the best counted strategy. This improvement of sliding window will be a more accurate, because of increased window.

The sliding window needs to remember two things action value and which action it was. This means it creates a lot of redundancy information. We could improve slightly memory requirements. We create a singleton unit to memorize or values which will neglect the memory requirements caused by multiple selectors. This limitation can increase the overall capacity of simulations, but decreases the capacity of independent selectors, which can lead to inaccuracy and problems with a too small number of simulations in some selectors.

Chapter 6

Conclusion

We read multiple articles about MCTS in imperfect-information games. We reviewed them in the Section 3 and chose to experiment on two main modifications. These modifications were the nondeterministic UCT and the Sliding Window UCT. We then implemented the Sliding Window and modified the game theory software package, we described this in Section 4.2. We experimented on UCT and their modifications to observe a convergence in games.

In Section 4, we empirically analyzed a convergence of UCT in instances of short matrix games as well as sequential simultaneous move games of various complexity. We empirically analyzed a convergence in a big group of matrix games. We identified simple example games that prevent convergence of UCT to the Nash equilibrium. We used these games in an empirical analysis of convergence. We found similar games having a problem with convergence. This problem was caused by the synchronization. We used gamut for creating randomly generated games. We create a lot of these games and we found one game which had a problem with convergence. We compared both modifications on this game. We analyzed sequential simultaneous move games in Section 4.6. In the first game, we compared different epsilons and showed that the nondeterministic modification of UCT depends heavily on an epsilon. In the second game, we focused on a convergence problem when one player has the Nash equilibria strategy similar to a pure strategy. In the third game, we tested and observed the sliding window in a simultaneous game.

We were unable to identify properties of games which will accelerate the converges, but in general if a game has the Nash equilibrium made of pure strategies UCT will converge very fast to it. The significant attribute of a rate of a convergence is also a distance between the lowest action value and the game value. The wrong behaviour was mainly observed in games where UCT changes strategy often, this doesn't apply for the sliding window. So if the strategy profile isn't stable it means it is very probably wrong.

The biggest difference is how many iterations are we going to use, this is essential. If we are planning to use a lot of simulations better is to use the non-deterministic modification of UCT, but if it will have a small restricted number of simulations is better to use sliding window and pick the best strategy it creates. The biggest downfall of the sliding window is that it will not remain in a strategy profile near Nash equilibria. So we need to watch the exploitability and remember the best exploitability, which is hard in big games. Basically, it will not always compute a better strategy with an increased number of iterations. So in

most cases are easier to use the non-deterministic modification of UCT and gave it more time to compute a strategy.

We discussed some possible improvements and were able to propose an improvement of the non-deterministic modification in Section 5. It increases the number of constant used by UCT from two to four. This can be very tricky, setting right constants can take some time. It is only slightly, but it can increase the chance and the speed of a convergence.

Bibliography

[1] Hassen Doghmen Adrien Couetoux and Olivier Teytaud. Improving the Exploration in Upper Confidence Trees. *LION 6*, page 366–371, 2012.

[2] Olivier Teytaud Adrien Couetoux, Jean-Baptiste Hoock, Nataliya Sokolovska and Nicolas Bonnard. Continuous Upper Confidence Trees. *LION 5*, page 433–445, 2011.

[3] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(1):1–43, 2012.

[4] Commons.wikimedia.org. File:mcts (english).svg - wikimedia commons, 2013.

[5] Peter I Cowling, Edward J Powley, and Daniel Whitehouse. Information set monte carlo tree search. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(2):120–143, 2012.

[6] Peter I Cowling, Edward J Powley, and Daniel Whitehouse. Information set monte carlo tree search. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(2):120–143, 2012.

[7] Adrien Couetoux David Auger and Olivier Teytaud. Bandit based monte-carlo planning. *ECML 2006*, page 282–293, 2006.

[8] Adrien Couetoux David Auger and Olivier Teytaud. Continuous upper confidence trees with polynomial exploration – consistency. *ECML PKDD 2013*, page 194–209, 2013.

[9] Michael Bowling Michael Johanson, Kevin Waugh and Martin Zinkevich. Accelerating Best Response Calculation in Large Extensive Games. *IJCAI'11 Proceedings of the Twenty-Second international joint conference on Artificial Intelligence*, 1:258–265, 2011. ISBN: 978-1-57735-513-7.

[10] Eugene Nudelman, Jennifer Wortman, Yoav Shoham, and Kevin Leyton-Brown. Run the gamut: A comprehensive approach to evaluating game-theoretic algorithms. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 880–887. IEEE Computer Society, 2004.

[11] Marc Ponsen, Steven De Jong, and Marc Lanctot. Computing approximate nash equilibria and robust best-responses using sampling. *Journal of Artificial Intelligence Research*, pages 575–605, 2011.

[12] Mohammad Shafiei, Nathan Sturtevant, and Jonathan Schaeffer. Comparing uct versus cfr in simultaneous games. 2009.

[13] Olivier Teytaud and Sébastien Flory. Upper Confidence Trees with Short Term Partial Information. *EvoApplications 2011*, page 153–162, 2011.

[14] Kevin Leyton-Brown Yoav Shoham. *MULTIAGENT SYSTEMS Algorithmic, Game-Theoretic, and Logical Foundations*. Masfoundation, Standford, USA, 2010. `http://www.masfoundations.org/mas.pdf`.

Appendix A

CD Content

Following folders are placed on CD accompanying this thesis

- Thesis source files Folder contains sources for this thesis. Can be compiled with pdflatex of attached TexLive bundle.
- Experiment results Contain all experimental data used in observations.
- Java code Contains game theoretical software.
- Matlab scripts Attached Matlab scripts used for graph generation.
- References Folder containing all publicly accessible papers used in this thesis.